In [1]:
```
from zipfile import ZipFile
import os

zip_file_path ="C:/women-fashion.zip"
extraction_directory = 'C:/women_fashion.zip'
if not os.path.exists(extraction_directory):
    os.makedirs(extraction_directory)

with ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extraction_directory)

extracted_files = os.listdir(extraction_directory)
print(extracted_files[:10])
```

```
['women fashion', '__MACOSX']
```

In [2]:
```
# correcting the path to include the 'women fashion' directory and listing
extraction_directory_updated = os.path.join(extraction_directory, 'women fa

# list the files in the updated directory
extracted_files_updated = os.listdir(extraction_directory_updated)
extracted_files_updated[:10], len(extracted_files_updated)
```

Out[2]:
```
(['.DS_Store',
  'anarkali suit with a long, olive green kurta adorned with intricate emb
roidery around the neckline and cuffs, paired with matching fitted trouser
s.jpg',
  'Anarkali suit with a modern twist.jpg',
  'Anarkali suit with fitted bodice with a high neckline.jpg',
  'anarkali suit with intricate silver embellishments on the neckline, sle
eves.jpg',
  'anarkali suit with lavender in color with intricate white patterns thro
ughout the fabric.jpg',
  'anarkali suit. It consists of a turquoise skirt with detailed golden em
broidery, a multicolored blouse with floral patterns, and an orange dupatt
a with lace borders.jpg',
  'ark green, knee-length dress with short sleeves and a white, patterned
neckline.jpg',
  'beige top adorned with black dots and a green skirt.jpg',
  'black and white gingham checkered A-line dress with a flared skirt.jp
g'],
 97)
```

In [3]:
```
from PIL import Image
import matplotlib.pyplot as plt

# function to load and display an image
def display_image(file_path):
    image = Image.open(file_path)
    plt.imshow(image)
    plt.axis('off')
    plt.show()

# display the first image to understand its characteristics
first_image_path = os.path.join(extraction_directory_updated, extracted_fil
display_image(first_image_path)
```

```
---------------------------------------------------------------------------
-
UnidentifiedImageError                    Traceback (most recent call las
t)
Input In [3], in <cell line: 13>()
     11 # display the first image to understand its characteristics
     12 first_image_path = os.path.join(extraction_directory_updated, extr
acted_files_updated[0])
---> 13 display_image(first_image_path)

Input In [3], in display_image(file_path)
      5 def display_image(file_path):
----> 6     image = Image.open(file_path)
      7     plt.imshow(image)
      8     plt.axis('off')

File ~\anaconda3\lib\site-packages\PIL\Image.py:3305, in open(fp, mode, fo
rmats)
   3303     warnings.warn(message)
   3304 msg = "cannot identify image file %r" % (filename if filename else
fp)
-> 3305 raise UnidentifiedImageError(msg)

UnidentifiedImageError: cannot identify image file 'C:/women_fashion.zip
\\women fashion\\.DS_Store'
```

In [ ]:
```
import glob

# directory path containing your images
image_directory = '/content/women_fashion/women fashion'

image_paths_list = [file for file in glob.glob(os.path.join(image_directory

# print the list of image file paths
print(image_paths_list)
```

In [ ]:
```
pip install TensorFlow
import TensorFlow
```

In [ ]:

```
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.models import Model
import numpy as np

base_model = VGG16(weights='imagenet', include_top=False)
model = Model(inputs=base_model.input, outputs=base_model.output)

def preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array_expanded = np.expand_dims(img_array, axis=0)
    return preprocess_input(img_array_expanded)

def extract_features(model, preprocessed_img):
    features = model.predict(preprocessed_img)
    flattened_features = features.flatten()
    normalized_features = flattened_features / np.linalg.norm(flattened_fea
    return normalized_features

all_features = []
all_image_names = []

for img_path in image_paths_list:
    preprocessed_img = preprocess_image(img_path)
    features = extract_features(model, preprocessed_img)
    all_features.append(features)
    all_image_names.append(os.path.basename(img_path))
```

In [ ]:
```
from scipy.spatial.distance import cosine

def recommend_fashion_items_cnn(input_image_path, all_features, all_image_n
    # pre-process the input image and extract features
    preprocessed_img = preprocess_image(input_image_path)
    input_features = extract_features(model, preprocessed_img)

    # calculate similarities and find the top N similar images
    similarities = [1 - cosine(input_features, other_feature) for other_fea
    similar_indices = np.argsort(similarities)[-top_n:]

    # filter out the input image index from similar_indices
    similar_indices = [idx for idx in similar_indices if idx != all_image_n

    # display the input image
    plt.figure(figsize=(15, 10))
    plt.subplot(1, top_n + 1, 1)
    plt.imshow(Image.open(input_image_path))
    plt.title("Input Image")
    plt.axis('off')

    # display similar images
    for i, idx in enumerate(similar_indices[:top_n], start=1):
        image_path = os.path.join('/content/women_fashion/women fashion', a
        plt.subplot(1, top_n + 1, i + 1)
        plt.imshow(Image.open(image_path))
        plt.title(f"Recommendation {i}")
        plt.axis('off')

    plt.tight_layout()
    plt.show()
```

In [ ]:
```
input_image_path = '/content/women_fashion/women fashion/dark, elegant, sle
recommend_fashion_items_cnn(input_image_path, all_features, image_paths_lis
```

In [4]:
```
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import plotly.io as pio
pio.templates.default = "plotly_white"

metro_data = pd.read_csv("C:/Users/qumrul hoda/Downloads/Delhi-Metro-Networ

print(metro_data.head())
```

```
   Station ID         Station Name  Distance from Start (km)          Line
\
0           1             Jhil Mil                      10.3      Red line
1           2  Welcome [Conn: Red]                      46.8     Pink line
2           3          DLF Phase 3                      10.0   Rapid Metro
3           4           Okhla NSIC                      23.8  Magenta line
4           5           Dwarka Mor                      10.2     Blue line

  Opening Date Station Layout   Latitude  Longitude
0   2008-04-06       Elevated  28.675790  77.312390
1   2018-10-31       Elevated  28.671800  77.277560
2   2013-11-14       Elevated  28.493600  77.093500
3   2017-12-25       Elevated  28.554483  77.264849
4   2005-12-30       Elevated  28.619320  77.033260
```

In [5]:
```
# checking for missing values
missing_values = metro_data.isnull().sum()

# checking data types
data_types = metro_data.dtypes

missing_values
```

Out[5]:
```
Station ID                  0
Station Name                0
Distance from Start (km)    0
Line                        0
Opening Date                0
Station Layout              0
Latitude                    0
Longitude                   0
dtype: int64
```

In [6]:
```
# converting 'Opening Date' to datetime format
metro_data['Opening Date'] = pd.to_datetime(metro_data['Opening Date'])
```

In [*]:
```
pip install folium
```

In [*]:
```
import folium
# defining a color scheme for the metro lines
line_colors = {
    'Red line': 'red',
    'Blue line': 'blue',
    'Yellow line': 'beige',
    'Green line': 'green',
    'Voilet line': 'purple',
    'Pink line': 'pink',
    'Magenta line': 'darkred',
    'Orange line': 'orange',
    'Rapid Metro': 'cadetblue',
    'Aqua line': 'black',
    'Green line branch': 'lightgreen',
    'Blue line branch': 'lightblue',
    'Gray line': 'lightgray'
}

delhi_map_with_line_tooltip = folium.Map(location=[28.7041, 77.1025], zoom_

# adding colored markers for each metro station with line name in tooltip
for index, row in metro_data.iterrows():
    line = row['Line']
    color = line_colors.get(line, 'black')  # Default color is black if lin
    folium.Marker(
        location=[row['Latitude'], row['Longitude']],
        popup=f"{row['Station Name']}",
        tooltip=f"{row['Station Name']}, {line}",
        icon=folium.Icon(color=color)
    ).add_to(delhi_map_with_line_tooltip)

# Displaying the updated map
delhi_map_with_line_tooltip
```

In [*]:
```
metro_data['Opening Year'] = metro_data['Opening Date'].dt.year

# counting the number of stations opened each year
stations_per_year = metro_data['Opening Year'].value_counts().sort_index()

stations_per_year_df = stations_per_year.reset_index()
stations_per_year_df.columns = ['Year', 'Number of Stations']

fig = px.bar(stations_per_year_df, x='Year', y='Number of Stations',
             title="Number of Metro Stations Opened Each Year in Delhi",
             labels={'Year': 'Year', 'Number of Stations': 'Number of Stati

fig.update_layout(xaxis_tickangle=-45, xaxis=dict(tickmode='linear'),
                  yaxis=dict(title='Number of Stations Opened'),
                  xaxis_title="Year")

fig.show()
```

In [*]:
```
stations_per_line = metro_data['Line'].value_counts()

# calculating the total distance of each metro line (max distance from star
total_distance_per_line = metro_data.groupby('Line')['Distance from Start (

avg_distance_per_line = total_distance_per_line / (stations_per_line - 1)

line_analysis = pd.DataFrame({
    'Line': stations_per_line.index,
    'Number of Stations': stations_per_line.values,
    'Average Distance Between Stations (km)': avg_distance_per_line
})

# sorting the DataFrame by the number of stations
line_analysis = line_analysis.sort_values(by='Number of Stations', ascendin

line_analysis.reset_index(drop=True, inplace=True)
print(line_analysis)
```

In [*]:
```
# creating subplots
fig = make_subplots(rows=1, cols=2, subplot_titles=('Number of Stations Per
                                                    'Average Distance Betwe
                    horizontal_spacing=0.2)

# plot for Number of Stations per Line
fig.add_trace(
    go.Bar(y=line_analysis['Line'], x=line_analysis['Number of Stations'],
           orientation='h', name='Number of Stations', marker_color='crimso
    row=1, col=1
)

# plot for Average Distance Between Stations
fig.add_trace(
    go.Bar(y=line_analysis['Line'], x=line_analysis['Average Distance Betwe
           orientation='h', name='Average Distance (km)', marker_color='nav
    row=1, col=2
)

# update xaxis properties
fig.update_xaxes(title_text="Number of Stations", row=1, col=1)
fig.update_xaxes(title_text="Average Distance Between Stations (km)", row=1

# update yaxis properties
fig.update_yaxes(title_text="Metro Line", row=1, col=1)
fig.update_yaxes(title_text="", row=1, col=2)

# update layout
fig.update_layout(height=600, width=1200, title_text="Metro Line Analysis",

fig.show()
```

In [*]:
```
layout_counts = metro_data['Station Layout'].value_counts()

# creating the bar plot using Plotly
fig = px.bar(x=layout_counts.index, y=layout_counts.values,
             labels={'x': 'Station Layout', 'y': 'Number of Stations'},
             title='Distribution of Delhi Metro Station Layouts',
             color=layout_counts.index,
             color_continuous_scale='pastel')

# updating layout for better presentation
fig.update_layout(xaxis_title="Station Layout",
                  yaxis_title="Number of Stations",
                  coloraxis_showscale=False,
                  template="plotly_white")

fig.show()
```

In [ ]:
```
```